Defect- and Variation-tolerant Logic Mapping in Nano-crossbar Using Bipartite Matching and Memetic Algorithm

Yuan, Bo; Li, Bin; Chen, Huanhuan; Yao, Xin

DOI:
10.1109/TVLSI.2016.2530898

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Yuan, B, Li, B, Chen, H & Yao, X 2016, 'Defect- and Variation-tolerant Logic Mapping in Nano-crossbar Using Bipartite Matching and Memetic Algorithm', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 9, pp. 2813-2826. https://doi.org/10.1109/TVLSI.2016.2530898

Defect- and Variation-Tolerant Logic Mapping in Nanocrossbar Using Bipartite Matching and Memetic Algorithm

Bo Yuan, *Member, IEEE*, Bin Li, *Member, IEEE*, Huanhuan Chen, *Senior Member, IEEE*, and Xin Yao, *Fellow, IEEE*

***Abstract*— High defect density and extreme parameter variation make it very difficult to implement reliable logic functions in crossbar-based nanoarchitectures. It is a major design challenge to tolerate defects and variations simultaneously for such architectures. In this paper, a method based on a bipartite matching and memetic algorithm is proposed for defect- and variation-tolerant logic mapping (D/VTLM) problem in crossbar-based nanoarchitectures. In the proposed method, the search space of the D/VTLM problem can be dramatically reduced through the introduction of the min–max weight maximum-bipartite-matching (MMW-MBM) and a related heuristic bipartite matching method. MMW-MBM is defined on a weighted bipartite graph as an MBM, where the maximal weight of the edges in the matching has a minimal value. In addition, a defect- and variation-aware local search (D/VALS) operator is proposed for D/VTLM and embedded in a global search framework. The D/VALS operator is able to utilize the domain knowledge extracted from problem instances and, thus, has the potential to search the solution space more efficiently. Compared with the state-of-the-art heuristic and recursive algorithms, and a simulated annealing algorithm, the good performance of our proposed method is verified on a 3-bit adder and a large set of random benchmarks of various scales.**

***Index Terms*— Fault tolerance, logic mapping, memetic algorithm (MA), nanoarchitecture, nanoelectronics.**

Manuscript received September 9, 2015; revised December 10, 2015; accepted February 1, 2016. Date of publication March 3, 2016; date of current version August 23, 2016. This work was supported in part by the Engineering and Physical Sciences Research Council under Grants EP/J017515/1 and EP/K001523/1; in part by the China Post-Doctoral Science Foundation under Grants 2014M560520 and 2015T80666; in part by the National Natural Science Foundation of China under Grants 61203292, 61329302, 61473271, 61503357, 61511130083, and 91546116; and in part by the Fundamental Research Funds for the Central Universities under Grant WK0110000046. The work of X. Yao was supported by the Royal Society Wolfson Research Merit Award. *(Corresponding author: Huanhuan Chen.)*

B. Yuan and H. Chen are with the School of Computer Science, UBRI, University of Science and Technology of China, Hefei 230026, China (e-mail: yuanbo@ustc.edu.cn; hchen@ustc.edu.cn).

B. Li is with UBRI and the Chinese Academy of Sciences Key Laboratory of Technology in Geo-Spatial Information Processing and Application System, University of Science and Technology (USTC), Hefei 230026, China (e-mail: binli@ustc.edu.cn).

X. Yao is with UBRI and the Centre of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2016.2530898

NOMENCLATURE

G_1	Weighted bipartite graph of crossbar architecture.
G_2	Bipartite graph of logic function.
N	Population size.
P	Parents.
B	Offspring.
t	Iteration counter.
f	Fitness value.
λ	Greedy strength factor.
P_{cross}	Probability of crossover.
P_{mut}	Probability of mutation.
P_{ls}	Probability of local search.

I. INTRODUCTION

NANOELECTRONICS [1], [2] has emerged with the hope of extending Moore's law beyond CMOS in the long-term future. It is expected to achieve much higher device density and operation frequency than that of conventional CMOS technologies. Recently, the world's first programmable nanoprocessor consisting of programmable, nonvolatile nanowire transistor arrays (PNNTAs) [3] has been published. This paper represents an important breakthrough of logic circuits built from the bottom–up paradigm [4] and shows tremendous opportunities for future computing systems. However, the nanochips produced from both the bottom–up process [4] and nanoimprint techniques [5] are inherently prone to high defect density and extreme parameter variation. This is because of the extremely small size of the nanodevices and the difficulty in controlling the fabricating process precisely.

The exact level of defect density is still unknown, but it is reasonable that 1%–15% of the resources, e.g., wires, switches, transistors, and so on, on a nanochip will be defective [6]. The Quantum Science Research group at Hewlett-Packard fabricated an 8×8 crossbar architecture using molecular switches at the crosspoints by nanoimprint lithography, where 15% of the switches were defective [5]. The researchers at Harvard and MITRE characterized the threshold voltage values of nodes from the fabricated PNNTA structure in both active and inactive states. They found that only 86% of nodes in active state and 87% of nodes in inactive state met the voltage requirements [3].

Parameter variation, e.g., fluctuations in length, width, oxide thickness, flat-band conditions, and so on, impacts both conventional and emerging technologies [7]. As device scaling, some individual parts of the device are made up of fewer atoms. If merely a single atom is out of place, the device characteristics are significantly changed. Variations in the geometry of the devices induct serious performance variations of the circuits. For example, density variations in carbon nanotubes growth can compromise the reliability of carbon nanotube field-effect transistor (FET) and result in increased delay variations [8]. Another example is the fin FET (FinFET) device, it has been shown through practical measurements and theoretical formulations [9] that quantum effects have great impact on the performance of FinFET, while the body thickness primarily determines these effects.

As stated above, design process is significantly complicated due to the lack of determinacy; besides, it is expected to be wore as device scaling. To deal with such a high defect density and extreme parameter variation simultaneously, one promising design paradigm for logic function implementation on nanochips is the defect- and variation-tolerant logic mapping (D/VTLM): given a nanoarchitecture and a logic function to be implemented on it, find a mapping of the logic function to the architecture with consideration of defects and variations.

Without consideration of the variations, the defect-tolerant logic mapping (DTLM) problem is equivalent to the subgraph isomorphism problem (SIP): return a subgraph of the (bipartite) graph G_1 that is isomorphic to the (bipartite) graph G_2. SIP is a well-known NP-complete problem [10]. While considering the variations, the D/VTLM problem is an extended version of SIP, which can be defined as: return a subgraph of G_1 that is isomorphic to G_2, and the subgraph has a minimal cost (e.g., path delay) among all subgraphs of G_1 that are isomorphic to G_2.

A number of methods have been proposed to deal with the DTLM problem, such as the recursive algorithm [11] based on backtracking and pruning, as well as various heuristic algorithms [12]–[15]. However, the runtime of the recursive algorithm is acceptable only for small-scale problems due to the recursive nature, while all the heuristic algorithms rely on fixed heuristics that show strong bias in favor of only small set of problems. The D/VTLM problem is highly complicated due to the additional consideration of variations, not only a valid mapping should be found, but also the path delay should be optimized. A simulated annealing algorithm (SA) [16] was first suggested due to its capability of exploration. The SA method has good effort on variation tolerance, but its efficiency is poor due to the huge search space. Recently, a set of integer linear programming (ILP) formulations were introduced in [17], but the ILP-based method has good results only on small-scale problems.

The mapping flow proposed in [18] is an efficient trail for DTLM by using the divided and conquer strategy that maximum-bipartite-matching (MBM) is introduced to reduce the search space. Inspired by this, this paper proposes a new matching problem and method. Specifically, the min–max weight MBM (MMW-MBM) problem is defined in this paper for the first time, and a heuristic MMW-MBM method is also presented. MMW-MBM is defined on a weighted bipartite graph as an MBM, where the maximal weight of the edges in the matching has a minimal value among all MBMs of the given graph. A naive way to find the MMW-MBM is to find all MBMs in a given graph first, and then select the one whose maximal edge weight is minimal. Instead of using such an enumeration method, a heuristic method for the MMW-MBM problem is developed to improve the time efficiency. Based on the MMW-MBM model and the heuristic algorithm, the problem of D/VTLM can be transferred to a pin assignment optimization problem.

For real-world optimization problems, it is often effective to incorporate problem-specific knowledge into local search strategies, which are referred to as memes in the case of memetic algorithms (MAs) [19], [20]. This paper proposes one such operator, called defect- and variation-aware local search (D/VALS). The key idea is inherited from the greedy reassignment (GR) local search operators for DTLM [18] and VTLM [21], which is to reassign the values of parts of the individual by taking advantage of the greedy information extracted from the problems. However, instead of individually utilizing defect information [18] or variation information [21], the D/VALS operator is capable of utilizing the combined information of both defect and variation.

Based on MMW-MBM (model and method) and the D/VALS operator, an MA is constructed to optimize the logic mapping in the reduced search space of D/VTLM. The MA employs a genetic algorithm (GA) as global search and the D/VALS operator as local search. With an appropriate coordination, the MA can not only exhibit a good explorative ability as a population-based global search algorithm does but also deliver a good exploitive performance as a local search algorithm does. Compared with the state-of-the-art recursive [11] and heuristic [14] algorithms, and the SA method [16], the performance of the proposed method is testified and verified on a 3-bit adder and a large set of random benchmarks of various scales. Experiment results show that a good performance on efficiency and effectiveness can be obtained by the proposed method.

The novelty of this paper can be summarized as follows. First, MMW-MBM is defined in this paper for the first time to reduce the search space of the D/VTLM problem. Second, instead of the enumeration method, a heuristic method is developed to find a MMW-MBM efficiently. Third, the D/VALS operator is designed under the considerations of both defect and variation information, and embedded in an MA framework.

The rest of this paper is organized as follows. Section II introduces the problem background and definition. In Section III, the search space of D/VTLM is reduced by MMW-MBM model. The detail of the D/VALS operator and the MA is presented in Section IV. Experimental studies and comparisons are given in Section V. Section VI concludes this paper.

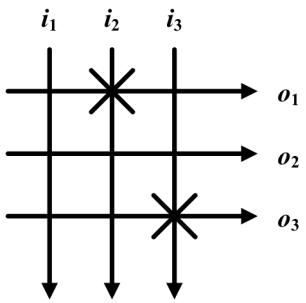

Fig. 1. 3×3 nanocrossbar with two defects.

II. Preliminaries

A. Nanocrossbar Architecture

A nanoelectronic crossbar consists of two layers of orthogonal nanowires. The region where two wires cross is called junction or crosspoint, which may be configured to implement a logic device. The assembly process has a stochastic nature that the probability of aligning three-terminal devices will be very low, while a two-terminal connection can be established more easily. Therefore, two-terminal devices, such as nanowire FETs, diodes, and molecular switches, are preferred [6].

In this paper, both the stuck-at-open defects and the stuck-at-close defects are considered. The stuck-at-open defect is representative of and most common in nanocrossbar architectures [22]. A stuck-at-open defect means that there is either a nonprogrammable switch or missing a switch at the crosspoint; thus, the two cross wires at this crosspoint are always disconnected. A stuck-at-close defect means that the switch at the crosspoint is permanently programmed, and the entire input wire and the output wire are unused. It is notable that the defect modeling of nanoelectronics is still an ongoing research problem. Without loss of generality, we may assume that the defects are independent and uniformly distributed as previous works did [23], [24]. This is a commonly employed assumption for theoretical research [25], which allows us to focus upon the essence of the proposed method instead of the physical details of the defects. It is notable that the approach presented in this paper can easily be extended to other defect types (nanowire open defect and nanowire bridging defect [26]) and other defect distributions (e.g., clustered distribution [27]) by modifying the following graph model slightly as discussed in [28].

An example of a defective 3×3 nanoelectronic crossbar is shown in Fig. 1. The crossbar consists of two sets of orthogonal nanowires. The vertical nanowires are the inputs (is), whereas the horizontal nanowires are the outputs (os). There is a programmable switch at each crosspoint. The nonprogrammable defective switches at the crosspoints are each represented by an X.

B. Problem Definition

A given 2-D crossbar with defects and delay variations can be represented by a weighted bipartite graph, as shown in Fig. 2. A weighted bipartite graph of an $m_1 \times m_2$ crossbar is an undirected weighted bipartite graph $G_1(I, O, C, W)$ with partitions I and O, having $|I| = m_1$ and $|O| = m_2$.

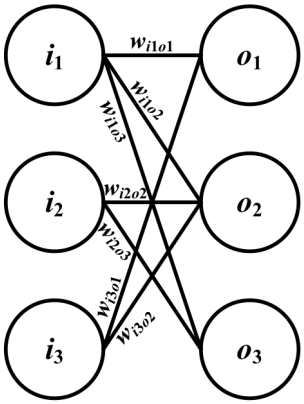

Fig. 2. Weighted bipartite graph of the nanocrossbar in Fig. 1.

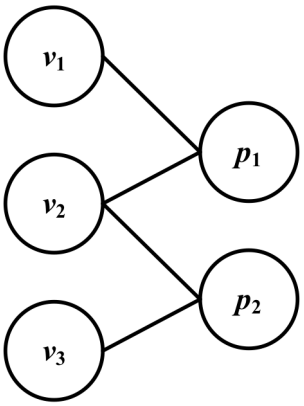

Fig. 3. Bipartite graph of logic function: $F = v_1v_2 + v_2v_3$.

I represents the set of input nanowires, and O represents the set of output nanowires. C consists of representative edges for all the programmable crosspoints in the crossbar. W is the set of delay variations correspond to the crosspoints.

A two-level logic function in a sum-of-products form can be represented by a bipartite graph $G_2(V, P, E)$, as shown in Fig. 3. In this scenario, V represents the set of logic variables, and P represents the set of product terms. E consists of representative edges for the corresponding product terms containing the variables.

When using a crossbar structure to implement a two-level logic function, the logical relationships between the variables and the product terms in the logic function can be represented by the connections between vertical and horizontal nanowires in the crossbar. Such logic-function-to-crossbar mapping problem can be formulated as an extended SIP: returning a subgraph of G_1 that is isomorphic to G_2, and the subgraph has a minimum cost (e.g., path delay) among all subgraphs of G_1 that are isomorphic to G_2.

The D/VTLM problem can be formally defined as the following. Given a defective $m_1 \times m_2$ crossbar weighted bipartite graph $G_1(I, O, C, W)$, and an $n_1 \times n_2$ logic function bipartite graph $G_2(V, P, E)$, find a node mapping M (M: $V \rightarrow I$, $P \rightarrow O$; $\forall(v, p) \in E$, $v \in V$, $p \in P$, $\exists(M(v), M(p)) \in C$) and $\text{Cost}(M) \leq \text{Cost}(M^*)$ for any node mapping M^* (M^*: $V \rightarrow I$, $P \rightarrow O$; $\forall(v, p) \in E$, $v \in V$, $p \in P$, $\exists(M^*(v), M^*(p)) \in C$).

The Cost(M) is the maximum path delay associated with the output of a crossbar after logic mapping. It is calculated in the proposed model as [17]

$$\text{Cost}(M) = \max_{\forall p \in P} \text{Cost}(M_p) \tag{1}$$

where Cost(M_p) represents the path delay associated with the product term p.

1) For FET-based nanocrossbar, the path delay of an output nanowire is decided by all activated crosspoints

$$\text{Cost}(M_p) = \sum_{v \in V \text{ and } e_{vp} \in E} W_{M(v)M(p)}. \tag{2}$$

2) For diode-based nanocrossbar, the path delay of an output nanowire is only decided by the activated crosspoint which has maximum delay

$$\text{Cost}(M_p) = \max_{v \in V \text{ and } e_{vp} \in E} W_{M(v)M(p)}. \tag{3}$$

III. Min–Max Weight Maximum-Bipartite-Matching

In fact, the mapping trail M consists of two mappings, one is $M(v)$: $V \to I$ and the other is $M(p)$: $P \to O$. Therefore, we can employ two decision vectors to represent the mapping trial M: input mapping vector (IMV) and output mapping vector (OMV).

1) IMV $[v] = i$, the vth variable is assigned to the ith input nanowire, $1 \le v \le n_1$, $1 \le i \le m_1$.
2) OMV $[p] = o$, the pth product term is mapped to the oth output nanowire, $1 \le p \le n_2$, $1 \le o \le m_2$.

It seems that we can search the whole solution space spanned by IMV and OMV as previous work did [16], but the extremely huge size of search space, $P(m_1, n_1) \times P(m_2, n_2)$, will make the problem very hard to be solved with limited computational resource, where $P(m, n)$ is the number of n-permutations of m. In order to solve the problem efficiently, the following parts will show that the problem can be solved in a divided and conquer way by introducing MMW-MBM, where IMV is optimized by a metaheuristic algorithm (Section IV) and OMV is determined by a heuristic algorithm (Section III).

A. Reducing the Search Space by MMW-MBM Model

As suggested in the previous works on DTLM [12], [14], when logic variables are previously assigned to input nanowires (IMV), the solution space of another mapping vector (OMV) will be restricted severely. For example, consider Figs. 2 and 3, if IMV is set as [1, 2, 3], which means v_1 is assigned to i_1, v_2 is assigned to i_2, and v_3 is assigned to i_3, thus p_1 cannot be assigned to o_1, because there is no edge between i_2 and o_1 in the crossbar bipartite graph. Therefore, we can construct a weighted bipartite graph to model which product terms can be assigned to which output nanowires and the corresponding cost (path delay), as shown in Fig. 4. While creating the weighted bipartite graph, we add one node on the left side for each product term p, and one node on the right side for each output nanowire o. An edge between p and o indicates that the product term p is compatible with the defect pattern of the crossbar, and can be realized by o. The associated weight is the path delay associated with the output o, which can be calculated according to (2) [or (3)]. For example, the weight between $p_1 = v_1 v_2$ and o_2 is $w_{\text{imv}(v1)o2} + w_{\text{imv}(v2)o2} = w_{i1o2} + w_{i2o2}$, because v_1 is assigned to i_1 and v_2 is assigned to i_2 indicated by the given IMV = [1, 2, 3].

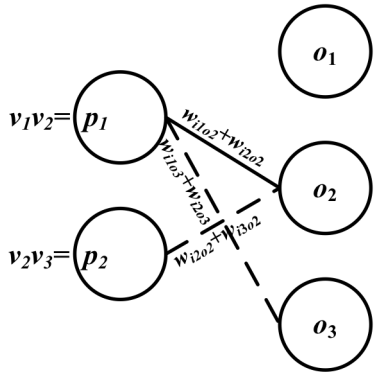

Fig. 4. Given IMV = [1, 2, 3], the weighted bipartite graph for product terms (Fig. 3) and output nanowires (Fig. 2).

Then, if we only consider defect tolerance [18], the problem is to find a complete assignment from the product terms to the output nanowires which is equivalent to the MBM problem [29]. Given an undirected bipartite graph $G = (U, V, E)$, where U and V are disjoint and all edges in E go between U and V. A matching is a subset of edges $Mat \in E$, such that for all vertices $v \in U \cup V$, at most one edge of Mat is incident on v. We say that a vertex $v \in U \cup V$ is matched by matching Mat if some edge in Mat is incident on v; otherwise, v is unmatched. A maximum matching is a matching of maximum cardinality, that is, a matching Mat such that for any matching Mat', we have $|Mat| \ge |Mat'|$. The set of dashed lines in Fig. 4 is an MBM in the graph. The problem MBM can be solved by Hungarian method or Ford–Fulkerson method [29].

However, if we consider defect tolerance and variation tolerance simultaneously, the problem is to find a complete assignment from the product terms to the output nanowires with minimum cost (1). So, the MBM problem needs to be extended to a new bipartite matching problem, the MMW-MBM problem. MMW-MBM can be defined on a weighted bipartite graph as an MBM, where the maximal weight of the edges in the matching has a minimal value among all MBMs of the given graph. One should note that the MMW-MBM problem is quite different from the maximum (minimum) weighted bipartite matching, which is defined on a complete weighted bipartite graph as a complete matching, where the sum of the weights of the edges in the matching has a maximal (minimal) value. A naive way to find the MMW-MBM is to find all MBMs in the given graph first, and then select the one whose maximal edge weight is minimal. Instead of using such an enumeration method, a heuristic method for MMW-MBM problem is presented to improve the efficiency.

Algorithm 1 Heuristic Method for MMW-MBM

```
//The proposed heuristic method for MMW-MBM
Input: Weighted bipartite graph: G = (U, V, E, W)
Output: MMW-MBM: Mat
 1: Mat ← Ford-Fulkerson Method for MBM (G)
 2: Sort E in descending order of their weights W
 3: Repeat
 4:    e ← edge in E with maximal weight
 5:    E ← E−{e}
 6:    Mat' ← Ford-Fulkerson Method for MBM (G)
 7:    If |Mat'| = |Mat| then
 8:       Mat = Mat'
 9:    Else
10:       Break
11:    End if
12: Until E = ϕ
13: Return Mat
```

Algorithm 2 Ford–Fulkerson Method for MBM [29]

```
//Ford-Fulkerson method for MBM
Input: Weighted bipartite graph: G = (U, V, E)
Output: MBM: Mat
1: Construct flow network G' = (V', E') based on G
2: Maximum flow f ← 0
3: While there exists an augmenting path p do
4:    Augment flow f along p
5: End while
6: Return Mat ← f
```

B. Heuristic MMW-MBM Method

Given an undirected weighted bipartite graph $G = (U, V, E, W)$, where U and V are disjoint and all edges in E go between U and V. Our heuristic is to remove the edges in G step by step in descending order of their weights, while an MBM algorithm (Ford–Fulkerson method) is used to check the cardinality of the current MBM until the cardinality reduces. The heuristic method is iterative, as shown in Algorithm 1. The algorithm starts with an initial matching *Mat* obtained by the Ford–Fulkerson method [29] (line 1), and then sort E in descending order according to their weights (line 2). At each iteration, the edge e in E with maximal weight is removed (lines 4 and 5), and then we obtain a new matching Mat' by running the Ford–Fulkerson method on the new graph G (line 6). If the cardinality of Mat' is equate to that of *Mat*, we update *Mat* to Mat' (line 8), otherwise, we break the loop (line 10) and then return *Mat* as the MMW-MBM (line 13). This process is repeated until E is empty.

Given a bipartite graph $G = (U, V, E)$, one can use the Ford–Fulkerson method [29] to find an MBM, as shown in Algorithm 2. The trick is to construct a flow network where the flow corresponds to matching. The corresponding flow network $G' = (V', E')$ for G is defined as follows. Let the source s and sink t be new vertices, and $V' = U \cup V \cup \{s, t\}$. The directed edges of G' are the edges of E, directed from U to V, along with $|U \cup V|$ new edges: $E' = \{(s, u): u \in U\} \cup \{(u, v): u \in U, v \in V, \text{ and } (u, v) \in E\} \cup \{(v, t): v \in V\}$. To complete the construction, unit capacity is assigned to each edge in E'. Thus, given an undirected bipartite graph G, one can find an MBM by creating the flow network G' (line 1), running the Ford–Fulkerson method (lines 2–5), and directly obtaining a maximum matching *Mat* from the integer-valued maximum flow f found (line 6). The Ford–Fulkerson algorithm starts with $f(u, v) = 0$ for all $u, v \in V'$, giving an initial flow of value 0 (line 2). At each iteration, the flow value is increased by finding an augmenting path that can be thought of simply as a path from the source s to the sink t along which more flow can be sent and augmented (lines 3–5). This process is repeated until no augmenting path can be found. The max-flow min-cut theorem proves that upon termination, this process yields a maximum flow.

Obviously, the heuristic method would return an MBM of the input graph, since the resulting matching *Mat* has the same cardinality as the initialized MBM obtained from the input graph (line 1 in Algorithm 1). Besides, the edges in G are removed according to the descending order of their weights, so the resulting matching *Mat* satisfies that the maximal weight of the edges in *Mat* has a minimal value among all MBMs of the input graph. Therefore, the heuristic method can indeed find an MMW-MBM in the given graph with the advantage of a high efficiency over the enumeration method.

Given a bipartite graph $G = (U, V, E)$, the time complexity of the Ford–Fulkerson Method is $O(|U \cup V||E|)$ [29]. Since the Ford–Fulkerson Method is used in the inner loop of the proposed heuristic method, it seems that the heuristic method would be very time-consuming. For example, one edge is to be removed from the graph in each loop, so the worst case time complexity of the heuristic method is $O(|U \cup V||E|^2)$. Fortunately, in the scenario of the D/VTLM problem, the graphs to be deal with by the heuristic method are highly sparse. If we assume that the edge density of the $m \times m$ crossbar bipartite graph G_1 is p, and the edge density of the $n \times n$ logic function graph G_2 is q, the probability that a product term can be realized by an output nanowire can be calculated as p^{qn} [12], which is the edge density of the input graph G in Algorithm 1. Therefore, the time complexity of the heuristic method is $O((m + n)(mnp^{qn})^2)$.

IV. Memetic Algorithm for D/VTLM

Given an IMV, the search space of OMV can be significantly reduced by creating the corresponding weighted bipartite graph modeling of which product terms can be assigned to which output nanowires and the corresponding path delay. Furthermore, it is possible to employ the proposed heuristic method to find an MMW-MBM exactly between product terms and output nanowires. Therefore, the next problem is how to choose an optimized IMV, so that the resulting MMW-MBM not only satisfies full defect tolerance (every product term corresponds to an output nanowire) but also exhibits good variation tolerance (minimized path delay). Due to the NP-hardness of the optimization of IMV, an MA is proposed. Besides incorporating an evolutionary computation framework to enhance the global optimization, the proposed MA gains pretty good performance by incorporating successful elements of previous effective greedy mapping algorithms.

Algorithm 3 MA for the D/VTLM

```
//The pseudo-code of MA for the D/VTLM
 1:  For i=1 to N do
 2:    P_i ← random permutation π
 3:    f(P_i) ← evaluate P_i according to Eq. 4, 5, and 6
 4:  End for
 5:  t = 0
 6:  Repeat
 7:    t = t+1
 8:    For i = 1 to N do
 9:      (P_j, P_k) ← Selection for Reproduction (P)
10:      B_i ← Crossover (P_j, P_k, P_cross)
11:    End for
12:    For i = 1 to N do
13:      B_i ← Mutation (B_i, P_mut)
14:      B_i ← D/VA Local Search (B_i, P_ls, λ)
15:    End for
16:    For i = 1 to N do
17:      f(B_i) ← evaluate B_i according to Eq. 4, 5, and 6
18:    End for
19:    P ← Selection for Survival (P, B, f)
20:  Until maximum runtime reached
```

A. Objective Function

Based on the obtained MMW-MBM, the following objective function can be defined for the given IMV:

$$\text{Objective} = \alpha \cdot dt + (1-\alpha) \cdot vt \tag{4}$$

$$dt = \sum_{p=1}^{P} m_p \cdot w_p / \sum_{p=1}^{P} w_p \tag{5}$$

$$vt = (\text{delay}_C - \text{delay}_M)/\text{delay}_C \tag{6}$$

where dt represents the capability of defect tolerance of the given IMV, while vt represents the capability of variation tolerance of the given IMV, and α is used to tune the impact of dt and vt on objective function. $m_p \in \{0, 1\}$ represents if product term p has a corresponding output nanowire o in the MMW-MBM under the given IMV, while weight w_p represents the impact of product term p on the dt value. delay_M represents the maximal weight of the edges in the MMW-MBM, while delay_C represents the maximal delay of the output nanowires in the crossbar.

Based on MMW-MBM (model and algorithm), the problem of D/VTLM is transferred to optimize the pin assignment from logic variables to input nanowires (IMV) with the evaluation by (4)–(6).

B. Framework of the MA

The outline of the proposed MA is given in Algorithm 3. A GA is used to work as the evolutionary computation framework of the MA due to its success history on many assignment problems [30]–[34]. The detailed design of the elementary steps of the algorithm is introduced as follows.

The algorithm starts with an initial population of N (population size) random individuals (line 2). Each random individual solution is evaluated according to (4)–(6) (line 3). The encoding of IMV solutions used in the implementation is straightforward. We encode the permutation π (denotes a permutation of the set $M = \{1, 2 \ldots m\}$) as a vector of input nanowires, such that the value j of the ith component in the vector indicates that the input nanowire j is assigned to logic variable i ($\pi(i) = j$). It is notable that the logic function size n is smaller than the crossbar architecture size m in some cases, thus IMV is an incomplete permutation. In order to take advantage of the off-the-shelf crossover operators, such as CX recombination [30], the complete permutation π is used instead of incomplete permutation. However, only the first n components will be decoded as IMV for the MMW-MBM-based fitness evaluation.

During every generation t, the population of N individuals generates N children through the crossover operator (line 10), the mutation operator (line 13), and the local search operator (line 14). The offspring is evaluated according to (4)–(6) (line 17) and then added to the current population. The CX recombination operator [30] has been testified to be an effective operator for assignment problems. It preserves the information contained in both parents in the sense that all alleles of the offspring are taken either from the first or from the second parent. The operator does not perform any implicit mutation, since an input nanowire j that is assigned to variable i in the child is also assigned to variable i in one or both parents. In the first phase, all input nanowires found at the same variable in the two parents are assigned to the corresponding variables in the offspring. Then, starting with a randomly chosen variable with no assignment, a nanowire is randomly chosen from the two parents. After that, additional assignments are made to ensure that no implicit mutation occurs. Then, the next unassigned variable to the right (in case we are at the end of the genome, we proceed at its beginning) is processed in the same way until all variables have been considered. Since the logic function size n may be smaller than or equal to the crossbar architecture size m, we consider applying a mutation operator in two cases.

1) If $n < m$, we will randomly select a gene within alleles $1 \sim n$ to be mutated and exchange its value with another gene from the last $m - n$ genes.
2) If $n = m$, we will randomly select two genes and then exchange their values. The local search operator will be explained in detail in Section IV-C.

Selection occurs two times in the main loop of the proposed MA. Selection for reproduction (line 9) is performed before a crossover operator can be applied, which is based on a purely random basis without bias to filter individuals, and selection for survival (line 19) is performed to reduce the population to its original size, which is achieved by choosing the best N individuals from the pool of parents and children [30].

C. Defect/Variation-Aware Local Search

The local search operator developed for the MA can be regarded as a type of knowledge-guided mutation. Given a parent chromosome, the operator produces a child chromosome that is expected to outperform the parent. The key idea of the operator is inherited from the previous GR local search operators for DTLM [18] and VTLM [21]. However,

instead of individually utilizing defect information [18] or variation information [21], the operator is capable of utilizing the combined information on both defect and variation. Thus, the proposed operator is called D/VALS here. There is good knowledge that has been testified to be effective on some instances for defect tolerance, that is, a more frequently used variable needs more functional crosspoints. By assigning the most frequently used variables in the product terms to the input nanowires with the smallest number of defects, the greedy assignment heuristic might find the feasible solution with a higher probability [13], [14]. In addition, for variation tolerance, an intuitive greedy knowledge is that a more frequently used variable should be assigned to an input nanowire with a minimal delay [21].

Since the operator is designed to be complementary to the stochastic search of GA, the incorporation of the operator should maintain the stochastic search. Besides, strong greediness will weaken the stochastic nature of global search, resulting in early convergence. Therefore, a control parameter is introduced to limit the elements (genes) of the given solution (parent chromosome) to be operated by the local search. For example, only $n_1 \cdot \lambda$ variables and their corresponding $n_1 \cdot \lambda$ input nanowires are randomly selected, where n_1 is the number of variables, and $0 \leq \lambda \leq 1$ is called greedy strength factor here.

In order to release the time overhead added to the iterative process of GA, the time complexity of the operator should be as low as possible. In fact, the attributes of variables and nanowires can be obtained in advance, such as the number of times to be included by product terms for each variable v, the number of functional crosspoints on each input nanowire i, and the path delay associated with each input nanowire i (under the assumption that all the functional crosspoints are active), they are marked as Degree(v), Degree(i), and Delay(i), respectively. The property of an input nanowire i is measured as: Property(i) $= \alpha \cdot$ Degree(i) $-$ Delay(i), to consider the defect-tolerant capability (more functional crosspoints) and variation-tolerant capability (less path delay) at the same time. Parameter α is used to make sure that the defect tolerance is the key task, and thus, its value is set as: $\alpha > \text{Max}_{\forall i} Delay(i)$. To sum up, the greedy information of the problem instance only needs to be extracted once before the optimization.

The outline of the proposed D/VALS is given in Algorithm 4. Given a pin assignment (IMV), $n_1 \times \lambda$ variables and their corresponding $n_1 \times \lambda$ input nanowires are randomly selected and remarked as unvisited (line 1). Then, a defect/variation-aware GR heuristic is applied on these selected variables and nanowires to get a new solution (IMV). If there are unvisited variables (line 2), we choose a variable v, whose Degree(v) is maximal (line 3), and a nanowire i, whose property is the best (line 4) and then assign i to v (line 5), and mark v and i as visited (line 6). When the list of unvisited variables is empty, we get the new pin assignment (line 8).

The importance of the D/VALS operator is as follows.

1) Compared with previous local search methods (such as the 2-opt [30] and the fast-2-opt [30] heuristics) that can be commonly used for combinatorial optimization problems, D/VALS is problem-specific and much more efficient. For the 2-opt and the fast-2-opt heuristics, in one local search process, a number of solutions are generated by performing random swapping. Thus, frequent quality evaluations are required to gain information for guiding search, and the MMW-MBM-based evaluation is time-consuming. While, for D/VALS, in one local search process, only one solution is generated by using the greedy information.
2) The greedy strength factor λ provides a flexible control on the randomness or greediness of the operator. The randomness/greediness of the operator will decrease/increase along with the increasing of λ. When $\lambda = 1$, the whole IMV will be operated according to the GR heuristic. In order to coordinate the statistic search of GA, the factor λ should be given a very small value ($\lambda = 0.1$ in our scenario). Although only a small part of the given solution is operated by the GR, the quality of the new generated solution will be improved with a high probability. For example, for random generated solutions, their fitness (4) is improved with an average probability of 70%$\sim$80% by performing D/VALS in our experiments on large-scale benchmarks.
3) The following experiments (Section V) will show the advantages of introducing the D/VALS operator to the global optimization.

Algorithm 4 Defect/Variation-Aware Local Search

// Defect- and variation-aware local search for pin assignment
Input: Pin assignment: *IMV*
Output: New pin assignment: *IMV*
1: Randomly select $n_1 \bullet \lambda$ variables and their corresponding $n \bullet \lambda$ input nanowires, mark them unvisited
2: **While** there are unvisited logic variables **do**
3: Find the unvisited variable v with maximum ***Degree(v)***
4: Find the unvisited input nanowire i with maximum ***Property(i)***
5: $IMV[v] = i$
6: Mark v and i as visited
7: **End while**
8: **Return** *IMV*

V. Experimental Studies

A. Parameter Setting

In objective function (4), parameter α is set a big value, $\alpha = 0.8$, since defect tolerance is the primary task. As suggested in the previous work [18], the value of weight w_p is related to the number of variables v_p in product term p, that is, it is harder to map a produce term p, whose v_p is larger. Therefore, w_p is set as v_p^4 experimentally. There is no difference between FET-based nanocrossbar (2) and diode-based nanocrossbar (3) from the perspective of the mapping algorithms, and the comparisons between different algorithms are consistent for both cases, so we record the former (2) in the experiment section to save space.

TABLE I
EXPERIMENTAL RESULTS OF THE HMA [14], THE RMA [11], THE SA [16], AND THE MA ON A 3-bit ADDER ($p_o = 10\%$ AND $p_c = 0.1\%$)

No.	HMA [14]			RMA [11]			SA [16]			MA		
	Psucc	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*
1	100%	0.012	254.3	65%	0.138	263.0	75%	0.134	219.0	100%	0.444	207.0
2	100%	0.010	299.7	95%	0.135	279.7	95%	0.063	222.4	100%	0.828	209.5
3	100%	0.010	279.2	80%	0.137	273.9	100%	0.098	218.5	100%	0.634	206.7
4	100%	0.010	266.9	95%	0.213	277.0	95%	0.081	216.0	100%	0.630	209.2
5	100%	0.010	284.6	80%	0.137	278.0	100%	0.073	213.6	100%	0.658	207.3
6	100%	0.015	272.9	100%	0.715	268.1	95%	0.085	215.1	100%	0.615	208.4
7	100%	0.010	289.6	90%	3.079	262.7	0%	NA	NA	100%	4.288	214.8
8	100%	0.010	252.4	100%	0.134	271.1	100%	0.065	216.7	100%	0.781	206.9
9	100%	0.010	271.8	75%	0.457	272.7	0%	NA	NA	100%	5.208	221.4
10	100%	0.010	280.6	55%	0.165	267.4	0%	NA	NA	95%	3.912	218.3
11	100%	0.010	280.4	90%	1.862	272.6	100%	0.078	222.4	100%	0.675	213.6
12	100%	0.010	282.1	95%	0.134	274.4	100%	0.068	216.4	100%	0.759	209.7
13	100%	0.010	290.2	80%	0.135	270.6	95%	0.069	215.7	100%	0.817	204.4
14	100%	0.010	255.2	55%	2.421	268.4	0%	NA	NA	100%	4.179	217.7
15	100%	0.048	254.4	75%	0.423	277.0	0%	NA	NA	95%	4.727	217.3
16	100%	0.010	283.4	60%	0.133	271.1	100%	0.063	221.2	100%	0.648	215.1
17	100%	0.010	290.6	85%	0.411	280.4	95%	0.093	227.4	100%	0.525	214.8
18	100%	0.010	276.5	90%	0.135	266.2	100%	0.069	213.7	100%	0.703	210.2
19	100%	0.010	261.4	55%	0.135	269.0	100%	0.075	220.9	100%	0.663	209.5
20	100%	0.010	291.9	55%	0.185	288.6	0%	NA	NA	100%	4.542	221.8

Since the computational complexity of the MMW-MBM-based fitness evaluation does not allow evolving large populations in reasonable time, the population size N is set $N = 10$ after testing N values from 2 to 20 experimentally. A large greedy strength factor λ will weaken the stochastic nature of evolutionary algorithm, thus we set $\lambda = 0.1$ empirically. We set optimal parameters $P_{\text{cross}} = 80\%$, $P_{\text{mut}} = 20\%$, and $P_{\text{ls}} = 100\%$ experimentally by cross-validation.

All the experiments in this paper are performed on a platform with two 2.33-GHz Intel Xeon Quad processors E5410 and 12G memory. However, all tested algorithms are implemented as monolithic processes, and no CPU core parallelism is exploited.

B. Case Study of 3-bit Adder

A 3-bit adder, as a widely used benchmark [23], [24], [26], is first used to test the performances of different algorithms. The adder is implemented by two-level logic in the sum-of-product form. It requires 16 input wires and 31 output wires for logic operations, with a minimum crossbar area of $16 \times 31 = 496$, and uses 147 crosspoints [24]. So, its logic density approximates to 30%. We attempted to map the 3-bit adder to 20 random generated 17×32 crossbar architectures with 10%–30% stuck-at-open defect density (p_o) and 0.1% stuck-at-close defect density (p_c). Delay variations of the crosspoints are generated by using a Gaussian distribution ($\mu = 50$ and $3\sigma = 30$) as [17] did.

The heuristic mapping algorithm (HMA) [14], the recursive mapping algorithm (RMA) [11], and the SA [16] are three representative algorithms for the DTLM and the D/VTLM whose performances have been testified successfully. Therefore, they are used for comparison here. We use a cutoff time of 10, 20, and 30 s for the SA and the proposed MA when the stuck-at-open defect densities are 10%, 20%, and 30%. Since the RMA is a recursive algorithm, we use a four times cutoff time, 40, 80, and 120 s. The HMA uses greedy pin assignment and incomplete graph construction strategies, so it is always the fastest one. All the algorithms are run independently for 20 times on each test instance. Tables I–III record the simulation results of different algorithms including the following.

1) *Psucc:* The success rate of the algorithm, i.e., the fraction of the 20 runs that found a valid mapping.
2) *AvgT:* The average runtime (in seconds) of the algorithm if it finds valid mappings in 20 runs.
3) *AvgD:* The average path delay (Delay_M) of the mapping if the algorithm finds valid mappings in 20 runs.

It is notable that the HMA is a deterministic algorithm, so the same result will be obtained after being run multiple times. Besides, the HMA and the RMA are proposed only for defect tolerance, so they cannot provide mappings with optimized path delay.

Table I shows the results when the stuck-at-open defect density of crossbars is 10%. It can be seen that the following holds.

1) The HMA has a success rate of 100% on all test instances.
2) The RMA can find valid mappings on all instances, but the success rate is relatively low (<60%) on several test instances (4 out of 20).
3) The SA fails on several test instances (6 out of 20), but has high success rate on other instances.
4) The MA has a success rate of 100% on most test instances (18 out of 20).
5) The runtime of these algorithms is acceptable.
6) Compared with the SA, the path delay is slightly reduced by the MA.

TABLE II

EXPERIMENTAL RESULTS OF THE HMA [14], THE RMA [11], THE SA [16], AND THE MA ON A 3-bit ADDER ($p_o = 20\%$ AND $p_c = 0.1\%$)

No.	HMA [14]			RMA [11]			SA [16]			MA		
	Psucc	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*
1	100%	0.095	247.4	50%	8.462	257.4	40%	0.288	228.1	100%	0.319	208.7
2	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
3	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
4	100%	0.011	269.1	0%	NA	NA	0%	NA	NA	100%	2.764	218.0
5	100%	0.010	252.3	20%	5.957	277.7	10%	0.336	256.7	100%	0.205	211.1
6	100%	0.065	253.2	20%	7.054	269.4	15%	0.437	240.2	100%	0.199	209.3
7	100%	0.015	276.9	50%	12.777	268.5	40%	0.274	235.1	100%	0.187	212.8
8	100%	0.010	274.7	45%	7.464	271.0	50%	0.253	245.2	100%	0.309	210.7
9	100%	0.179	271.4	10%	1.277	263.9	0%	NA	NA	100%	2.749	224.0
10	100%	0.010	265.0	15%	13.295	258.9	0%	NA	NA	100%	4.532	213.2
11	100%	0.010	251.7	25%	13.234	272.9	0%	NA	NA	100%	1.770	219.3
12	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
13	100%	0.030	272.6	15%	0.143	279.1	0%	NA	NA	100%	3.285	212.7
14	100%	0.081	248.5	15%	1.598	270.7	0%	NA	NA	100%	2.324	217.0
15	100%	0.034	248.4	70%	7.228	271.8	20%	0.351	239.3	100%	0.335	208.6
16	100%	0.050	286.1	0%	NA	NA	0%	NA	NA	100%	1.651	223.5
17	100%	0.189	309.1	5%	0.159	280.0	0%	NA	NA	100%	3.238	228.2
18	100%	0.010	259.5	40%	0.381	271.9	45%	0.237	237.3	100%	0.310	207.3
19	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	4.280	219.6
20	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA

TABLE III

EXPERIMENTAL RESULTS OF THE HMA [14], THE RMA [11], THE SA [16], AND THE MA ON A 3-bit ADDER ($p_o = 30\%$ AND $p_c = 0.1\%$)

No.	HMA [14]			RMA [11]			SA [16]			MA		
	Psucc	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*
1	100%	0.017	291.5	0%	NA	NA	0%	NA	NA	100%	0.243	219.0
2	100%	0.106	271.1	5%	0.146	279.5	0%	NA	NA	100%	1.537	217.6
3	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	0.152	218.8
4	100%	0.010	261.7	0%	NA	NA	5%	1.942	244.9	100%	0.189	216.6
5	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	9.449	227.0
6	100%	0.044	278.2	0%	NA	NA	0%	NA	NA	100%	2.331	218.6
7	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	5.800	218.6
8	100%	0.159	286.9	0%	NA	NA	0%	NA	NA	100%	2.251	223.0
9	100%	0.054	255.1	0%	NA	NA	0%	NA	NA	100%	0.125	220.1
10	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	3.597	223.5
11	100%	0.030	252.6	0%	NA	NA	0%	NA	NA	100%	0.115	220.7
12	100%	0.015	281.9	5%	80.3	266.4	5%	1.200	240.7	100%	0.163	213.2
13	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	0.231	211.2
14	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	1.275	229.1
15	100%	0.070	263.6	0%	NA	NA	0%	NA	NA	100%	0.152	216.4
16	100%	0.087	286.4	5%	14.4	275.1	5%	3.257	251.7	100%	0.136	227.8
17	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
18	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	3.367	225.2
19	0%	NA	NA	0%	NA	NA	0%	NA	NA	100%	3.947	218.9
20	100%	0.279	282.3	0%	NA	NA	0%	NA	NA	100%	1.181	228.0

Table II shows the results when the stuck-at-open defect density of crossbars is 20%. It can be seen that the following holds.

1) The HMA has a success rate of 100% on most test instances (15 out of 20).
2) The RMA can find valid mappings on more than half of the instances (13 out of 20), but the success rate is very low.
3) The SA fails on most test instances (13 out of 20), and has low success rate (<50%) on other instances.
4) The MA has a success rate of 100% on most test instances (16 out of 20).
5) The runtime of these algorithms is still acceptable, although the runtime of the RMA increases a lot.
6) Compared with the SA, the path delay is significantly reduced by the MA.

Table III shows the results when the stuck-at-open defect density of crossbars is 30%. It can be seen that the following holds.

1) The HMA has a success rate of 100% on nearly half of the test instances (11 out of 20).

TABLE IV
EXPERIMENTAL RESULTS OF THE HMA [14], THE RMA [11], THE SA [16], AND THE MA ON RANDOM BENCHMARKS OF SIZE 16 × 16

No.	HMA [14]			RMA [11]			SA [16]			MA		
	Psucc	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*
1	100%	0.005	577.7	55%	1.964	536.6	100%	0.092	**413.5**	100%	**0.055**	420.0
2	100%	0.005	464.8	40%	3.303	494.9	100%	0.106	383.6	100%	**0.033**	**382.0**
3	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
4	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
5	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
6	100%	0.011	437.0	65%	5.862	453.4	100%	0.111	**366.3**	100%	**0.028**	369.3
7	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
8	100%	0.003	546.1	85%	0.446	551.6	100%	0.065	474.5	100%	**0.008**	**463.4**
9	100%	0.003	407.9	80%	1.998	457.5	100%	0.069	**366.0**	100%	**0.015**	367.8
10	100%	0.003	538.6	65%	8.403	556.7	100%	0.093	468.6	100%	**0.010**	**463.9**
11	100%	0.003	501.3	85%	0.373	461.2	100%	0.055	389.6	100%	**0.025**	**388.1**
12	100%	0.003	468.3	100%	1.005	457.1	100%	0.057	355.3	100%	**0.018**	**352.7**
13	100%	0.018	508.0	55%	4.881	485.2	100%	0.098	**415.2**	100%	**0.050**	419.7
14	100%	0.003	540.9	85%	1.090	556.8	100%	0.048	496.3	100%	**0.014**	**491.6**
15	100%	0.005	503.6	60%	2.879	470.4	100%	0.070	**368.0**	100%	**0.030**	380.0
16	100%	0.003	535.4	85%	1.583	584.8	100%	0.055	442.1	100%	**0.025**	442.3
17	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
18	100%	0.009	464.1	85%	6.613	458.6	100%	0.061	**393.4**	100%	**0.030**	395.9
19	100%	0.003	504.9	75%	1.497	500.9	100%	0.078	**422.8**	100%	**0.020**	425.8
20	100%	0.003	549.5	30%	9.182	541.7	100%	0.181	436.0	100%	**0.017**	**429.6**

2) The RMA and the SA can find valid mappings on few instances (3 out of 20), but the success rate approximates to zero (5%).
3) The MA has a success rate of 100% on most test instances (19 out of 20).
4) The runtime of these algorithms is still acceptable, except the runtime of the RMA.
5) Compared with the SA, the path delay is significantly reduced by the MA.

For the 3-bit adder, the above simulation results (Tables I–III) reveal that the HMA is a good choice for defect tolerance when the defect density is relatively low (10% or 20%). The RMA and the SA work well only in the case of low defect density (10%), and the RMA is very time-consuming as the defect density increases (30%). The MA is effective and efficient in all cases, and nearly 100% success rate can be achieved. Beside, compared with the SA, the MA can provide better optimizations on path delay.

C. Random Benchmark Instances

As can be seen from the above simulations, given fixed crossbar size and defect density, the results of the same algorithm are quite different on different crossbar architectures. This is because their defect patterns are different. In addition, this problem also exists for logic function that even if both the size and the logic density are fixed, the difficult of mapping different logic blocks is quite different due to the different logic patterns. To provide a sound and fair evaluation and comparison of different algorithms, a large set of benchmark graphs for logic functions and crossbar architectures are generated randomly as previous works did [13], [16]. For benchmark graphs of crossbar architectures, we set stuck-at-open defect density at 10% and stuck-at-close defect density defect density at 0.1%. Delay variations of the crosspoints (weights) are generated by using a Gaussian distribution ($\mu = 50$ and $3\sigma = 30$) as [17] did. For benchmark graphs of logic functions, we set logic density at 40%, a typical value as suggested in [16].

We use a cutoff time of 10, 20, and 60 s for the SA and the proposed MA when the logic function sizes are 16×16, 24×24, and 48×48. We attempt to map the logic functions to 20 random generated 16×16, 24×24, and 52×52 crossbar architectures. Since the RMA is a recursive algorithm, we use a four times cutoff time, 40, 80, and 240 s. The HMA uses greedy pin assignment and incomplete graph construction strategies, so it is always the fastest one. All the algorithms are run independently for 20 times on each test instance. Tables IV–VI record the simulation results of different algorithms including the following.

1) *Psucc:* The success rate of the algorithm, i.e., the fraction of the 20 runs that found a valid mapping.
2) *AvgT:* The average runtime (in seconds) of the algorithm if it finds valid mappings in 20 runs.
3) *AvgD:* The average path delay (Delay_M) of the mapping if the algorithm finds valid mappings in 20 runs.

It is notable that the HMA is a deterministic algorithm, so the same result will be obtained after being run multiple times. Besides, the HMA and the RMA are proposed only for defect tolerance, so they cannot provide mappings with optimized path delay.

We also perform statistical tests for the runtimes and path delays of paired evolutionary algorithms (EAs), the SA versus the MA, on each single benchmark instance. In particular, a two-tailed t-test is conducted with a null hypothesis stating that there is no difference between the two algorithms in comparison. The null hypothesis is rejected if the p-value is smaller than the significance level $\alpha = 0.05$. The runtime (or the path delay) of the algorithm, that is, statistically shorter than the other EA, will be highlighted in bold in tables.

TABLE V

EXPERIMENTAL RESULTS OF THE HMA [14], THE RMA [11], THE SA [16], AND THE MA ON RANDOM BENCHMARKS OF SIZE 24 × 24

No.	HMA [14]			RMA [11]			SA [16]			MA		
	Psucc	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*
1	100%	0.008	851.1	15%	14.624	782.7	100%	0.343	661.5	100%	**0.078**	**657.8**
2	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
3	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
4	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
5	100%	0.038	631.6	35%	7.047	676.3	100%	0.184	**545.6**	100%	**0.102**	562.9
6	0%	NA	NA	10%	6.042	782.1	100%	0.679	**651.3**	100%	**0.134**	663.1
7	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
8	100%	0.006	705.7	15%	20.801	746.9	100%	0.335	596.6	100%	**0.032**	**593.9**
9	0%	NA	NA	30%	9.661	794.4	100%	0.346	652.3	100%	**0.039**	**647.5**
10	100%	0.006	774.2	30%	4.273	797.8	100%	0.178	685.9	100%	**0.030**	**683.2**
11	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
12	0%	NA	NA	0%	NA	NA	100%	0.953	713.3	100%	**0.289**	**710.3**
13	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
14	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
15	100%	0.032	722.9	10%	0.544	688.9	100%	0.314	600.1	100%	**0.081**	604.8
16	100%	0.014	775.6	10%	2.598	729.4	100%	0.425	644.3	100%	**0.061**	**605.3**
17	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA
18	0%	NA	NA	10%	17.498	724.2	100%	0.865	**591.4**	100%	**0.551**	599.4
19	100%	0.006	768.1	45%	2.702	719.0	100%	0.215	630.5	100%	**0.034**	**627.9**
20	0%	NA	NA	0%	NA	NA	0%	NA	NA	0%	NA	NA

TABLE VI

EXPERIMENTAL RESULTS OF THE HMA [14], THE RMA [11], THE SA [16], AND THE MA ON RANDOM BENCHMARKS OF SIZE 48 × 48

No.	HMA [14]			RMA [11]			SA [16]			MA		
	Psucc	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*
1	0%	NA	NA	0%	NA	NA	5%	54.000	1195.6	100%	**12.721**	**1176.2**
2	0%	NA	NA	0%	NA	NA	0%	NA	NA	45%	42.553	1316.2
3	0%	NA	NA	0%	NA	NA	5%	24.645	1361.1	100%	**4.673**	**1268.5**
4	0%	NA	NA	0%	NA	NA	10%	27.357	1282.9	100%	**10.999**	**1277.7**
5	0%	NA	NA	0%	NA	NA	0%	NA	NA	85%	36.701	1235.4
6	0%	NA	NA	0%	NA	NA	0%	NA	NA	65%	35.771	1348.8
7	100%	0.067	1254.5	0%	NA	NA	85%	20.730	1153.5	100%	**3.525**	**1137.5**
8	100%	0.027	1384.2	0%	NA	NA	25%	30.223	1286.1	100%	**6.011**	**1250.1**
9	100%	0.428	1429.0	0%	NA	NA	45%	18.443	1221.6	100%	**5.535**	**1163.1**
10	0%	NA	NA	0%	NA	NA	25%	33.426	1229.4	100%	**7.076**	**1199.3**
11	0%	NA	NA	0%	NA	NA	0%	NA	NA	90%	34.249	1345.7
12	0%	NA	NA	0%	NA	NA	0%	NA	NA	25%	54.531	1245.2
13	0%	NA	NA	0%	NA	NA	25%	47.230	**1259.4**	95%	**27.234**	1290.6
14	0%	NA	NA	0%	NA	NA	20%	34.672	1159.7	100%	**5.800**	**1129.2**
15	100%	0.230	1445.5	0%	NA	NA	25%	37.365	1377.8	100%	**5.608**	**1279.6**
16	0%	NA	NA	0%	NA	NA	20%	36.164	1178.6	100%	**17.195**	**1162.4**
17	100%	0.119	1291.8	0%	NA	NA	75%	12.084	1158.1	100%	**0.900**	1157.3
18	0%	NA	NA	0%	NA	NA	0%	NA	NA	95%	28.663	1265.5
19	100%	0.260	1317.0	0%	NA	NA	95%	17.003	1173.0	100%	**2.133**	**1166.2**

Table IV shows the results when we map 16 × 16 logic functions to 16×16 crossbars. It can be seen that the following holds.

1) All the algorithms can find valid mapping on 15 test instances, and the HA, the SA, and the MA have a success rate of 100% on these test instances, while the RMA has relatively low success rate.
2) The runtime of these algorithms is acceptable, although the runtime of the RMA is several orders of magnitude of the other algorithms. It is evident that the MA is much faster than the SA.
3) There is no obvious difference between the SA and the MA from the viewpoint of path optimization.

Table V shows the results when we map 24 × 24 logic functions to 24×24 crossbars. It can be seen that the following holds.

1) The HA has a success rate of 100% on several test instances (7 out of 20), while the SA and the MA have a success rate of 100% on half of the test instances (11 out of 20).
2) Although the RMA can find valid mapping on half of the test instances (10 out of 20), the success rate is very low (<40%).
3) The runtime of these algorithms is acceptable, although the runtime of the RMA is several orders of magnitude of the other algorithms. It is evident that the MA is much faster than the SA.

4) There is no obvious difference between the SA and the MA from the viewpoint of path optimization.

As can be seen from Tables IV and V, all the algorithms fail on some test instances. One possible reason is that there is no valid solution at all. Another possible reason is that the granted runtime of the RMA, the SA, and the MA is not long enough to fully complete the search. Therefore, we can check it by using the enumeration method or grant a much longer runtime to the algorithms. Since it is not the main concern in this paper, the simulations are omitted to save space.

Table VI shows the results when we map 48 × 48 logic functions to 52 × 52 crossbars. It can be seen that the following holds.

1) The HA has a success rate of 100% on several test instances (6 out of 20).
2) The RMA fails on all the test instances, although granted a very long runtime (240 s).
3) The SA can find valid mappings on more than half of the test instances (14 out of 20), but it is unable to achieve a high success rate.
4) The MA has a success rate of 100% on more than half of the test instances (13 out of 20), and high success rate on other test instances (excluding No. 2 and No. 12).
5) Compared with the SA, the path delay is significantly reduced by the MA on most test instances.

We tried to map 48 × 48 logic functions to 48 × 48 crossbars, but all the algorithms failed on all test instances. As pointed in [35], given fixed defect density, the density of valid mappings increases with the crossbar size, so we consider a slightly larger size here, 52 × 52.

For random benchmarks, the above simulation results (Tables IV–VI) reveal that the HMA is a good choice for defect tolerance when the problem scale is relatively low (16 × 16 or 24 × 24). The RMA works only in the case of small-scale problem (16 × 16), and the RMA does not work on large-scale problem (48 × 48) even granted a very long runtime. The SA works well on the defect and variation tolerance when the problem scale is relatively low (16 × 16 or 24 × 24). The MA is the best, and nearly 100% success rate can be achieved on most test instances. Beside, compared with the SA, the MA is much faster and can provide better optimizations on path delay on large-scale problems (48 × 48).

It is very difficult to get a high success rate on the same test instances. We think that the reason is twofold. The first is the high computational complexity that the problem is NP-complete. In this case, we can do multiple searches to run the proposed algorithm more than one time, since the algorithm is a stochastic search in nature (rather than deterministic search), it will increase the chance to find a valid mapping. As shown in Fig. 5, the success rate increases significantly with the increase in the number of runs. On the other hand, it is possible that there is no valid mapping of the instance at all, so that we cannot find a valid mapping even using the enumeration method. In this case, we can use a larger crossbar to implement the function, or use multiple crossbars to partition the function.

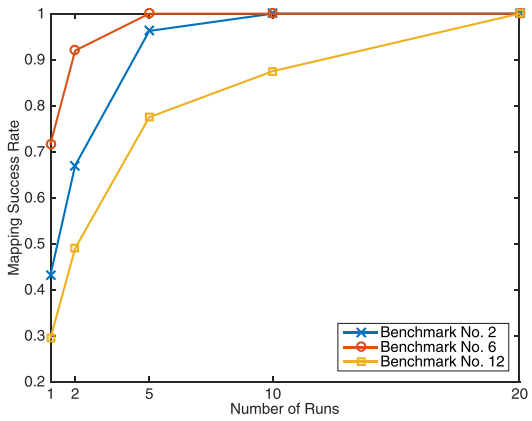

Fig. 5. Mapping success rate versus the number of runs on three test instances of size 48 × 48 (No. 2, No. 6, and No. 12 from Table VI).

TABLE VII
EXPERIMENTAL RESULTS OF THE GA AND THE MA ON RANDOM BENCHMARKS OF SIZE 48 × 48

No.	GA			MA		
	Psucc	*AvgT*	*AvgD*	*Psucc*	*AvgT*	*AvgD*
1	10%	30.744	1254.3	100%	**12.721**	**1176.2**
2	0%	NA	NA	45%	42.553	1316.2
3	65%	26.770	1408.7	100%	**4.673**	**1268.5**
4	55%	37.378	1361.4	100%	**10.999**	**1277.7**
5	0%	NA	NA	85%	36.701	1235.4
6	5%	**23.607**	1375.9	65%	35.771	**1348.8**
7	90%	15.195	1198.7	100%	**3.525**	**1137.5**
8	55%	20.864	1315.4	100%	**6.011**	**1250.1**
9	70%	18.963	1248.9	100%	**5.535**	**1163.1**
10	85%	26.319	1246.9	100%	**7.076**	**1199.3**
11	0%	NA	NA	90%	34.249	1345.7
12	0%	NA	NA	25%	54.531	1245.2
13	5%	43.442	1323.6	95%	**27.234**	**1290.6**
14	35%	29.310	1179.6	100%	**5.800**	**1129.2**
15	45%	23.403	1359.5	100%	**5.608**	**1279.6**
16	10%	43.790	1211.0	100%	**17.195**	**1162.4**
17	100%	0.901	1168.7	100%	0.900	**1157.3**
18	0%	NA	NA	95%	28.663	1265.5
19	100%	2.771	1188.3	100%	**2.133**	**1166.2**
20	40%	17.185	1181.5	100%	**9.442**	**1147.6**

D. Effectiveness of the Defect/Variation-Aware Local Search

In the proposed MA, the D/VALS operator is proposed to utilize the domain knowledge, so that the MA is expected to have the potential to search the solution space more efficiently. A very natural question is whether the proposed D/VALS operator has any positive contribution to the performance of the algorithm. To answer this question, we can remove it from the algorithm, while keeping all the other parts unchanged. Therefore, another evolutionary algorithm is added to the comparison, which is a GA following the flow of the MA, but without the D/VALS operator. The parameters of the GA are set as the same as the MA.

Table VII shows the results when we map 48 × 48 logic functions to 52 × 52 crossbars. It can be seen clearly that the incorporation of the D/VALS operator results in significantly enhanced results on all test instances. This is consistent with other results that demonstrated the advantage of using domain knowledge in evolutionary search [36]. Compared with the GA, *Psucc*, *AvgT*, and *AvgD* of the MA are improved a lot,

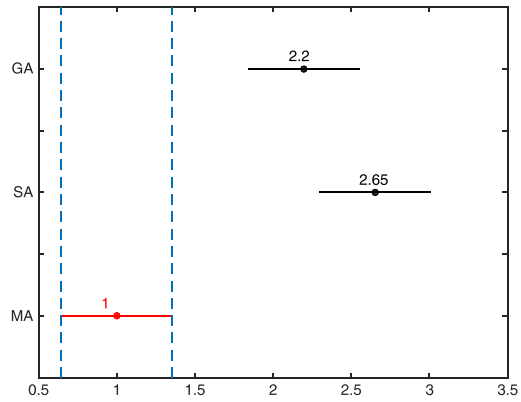

Fig. 6. Result of the Freidman test for comparing the performance of the EAs on 20 random benchmarks of size 48×48. The dots indicate the average ranks, the bars indicate the critical difference with the Bonferroni–Dunn test at significance level 0.05, and compared algorithms having nonoverlapped bars are significantly different.

even on the instances that are very difficult for the GA. The comparison between the GA and the MA demonstrates the advantages of introducing the D/VALS operator.

E. Statistical Comparisons Over Multiple Benchmarks

In Sections V. B–D, we have shown the performance of the algorithms (the SA, the GA, and the MA) on each independent benchmark instance. In order to statistically compare these algorithms based on multiple benchmark instances, we perform Freidman test [37], which is based on the ranks of compared algorithms. Freidman test in conjunction with Bonferroni–Dunn test [38] is used as post-hoc tests when all estimators are compared with the control estimator. The performance of pairwise comparison is significantly different if the corresponding average ranks differ by at least the critical difference

$$\mathrm{CD} = q_\alpha \sqrt{j(j+1)/6T} \tag{7}$$

where j is the number of algorithms ($j = 3$), T is the number of benchmark instances ($T = 20$) for a given problem scale, and critical values q_α can be found in [39]. For example, when $j = 3$, $q_{0.05} = 2.241$, where the subscript 0.05 is the significance level.

We rank the algorithms on *Psucc*, and record the ranking of each algorithm as 1, 2, and 3. Average ranks are assigned in the case of ties. The average rank of a single algorithm is obtained by averaging over all of data sets.

Fig. 6 shows the Friedman test results of the algorithms on large-scale problems. Since we employ the significance level 0.05, the critical difference is $\mathrm{CD} = 0.71$ with $j = 3$ and $T = 20$. It can be seen that the differences of the MA versus the SA and the MA versus the GA are greater than the critical difference, so the differences are significant, which means the MA is significantly better than the SA and the GA in these cases.

VI. Conclusion

As pointed in [6], although the dominant benefit of nanoelectronics is the enormous integration levels they may be able to achieve, one of the challenges for nanoelectronics is whether nanoscale devices can be reliably assembled into architectures. Some small-scale successes have been demonstrated, and the most promising architectures to date are crossbar-based [3], [5]. Reliability is a real challenge for nanoelectronics. It seems evident that the manufacturing techniques may never be able to produce perfect chips, so fault tolerance will be a key to the success of nanoelectronics. Another aspect of nanoelectronics that is quite different from current technologies is the electronic design automation (EDA) flow. The challenge is to deploy a circuit on a nanoelectronic chip when each chip is unique.

This paper contributes to EDA methods for the reliability design of nanocrossbar architectures. By introducing MMW-MBM, a new framework for solving the D/VTLM problem is proposed. MMW-MBM is a new matching problem that is defined here for the first time. In order to obtain an MMW-MBM solution efficiently, a heuristic method is presented. Furthermore, a new MA is proposed to implement the framework, in which a novel local search operator, D/VALS, is designed to make good use of the domain knowledge extracted from the problems. The performance of the proposed MA was evaluated on a 3-bit adder and a large set of random benchmarks. Our experimental results show that the D/VALS operator can help the algorithm to find near optimal solutions with a higher success rate and low computational resources. Compared with the state-of-the-art algorithms, the proposed MA algorithm has the advantage of getting a good balance between effectiveness and efficiency on various test instances.

References

[1] G. Bourianoff, J. E. Brewer, R. Cavin, J. A. Hutchby, and V. Zhirnov, "Boolean logic and alternative information-processing devices," *Computer*, vol. 41, no. 5, pp. 38–46, May 2008.

[2] R. Cavin, J. A. Hutchby, V. Zhirnov, J. E. Brewer, and G. Bourianoff, "Emerging research architectures," *Computer*, vol. 41, no. 5, pp. 33–37, May 2008.

[3] H. Yan *et al.*, "Programmable nanowire circuits for nanoprocessors," *Nature*, vol. 470, pp. 240–244, Feb. 2011.

[4] W. Lu and C. M. Lieber, "Nanoelectronics from the bottom up," *Nature Mater.*, vol. 6, no. 11, pp. 841–850, 2007.

[5] Y. Chen *et al.*, "Nanoscale molecular-switch crossbar circuits," *Nanotechnology*, vol. 14, no. 4, pp. 462–468, 2003.

[6] M. Haselman and S. Hauck, "The future of integrated circuits: A survey of nanoelectronics," *Proc. IEEE*, vol. 98, no. 1, pp. 11–38, Jan. 2010.

[7] S. Ghosh and K. Roy, "Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era," *Proc. IEEE*, vol. 98, no. 10, pp. 1718–1751, Oct. 2010.

[8] J. Zhang, N. Patil, A. Hazeghi, and S. Mitra, "Carbon nanotube circuits in the presence of carbon nanotube density variations," in *Proc. 46th ACM/IEEE Design Autom. Conf.*, Jul. 2009, pp. 71–76.

[9] S. Xiong and J. Bokor, "Sensitivity of double-gate and FinFET devices to process variations," *IEEE Trans. Electron Devices*, vol. 50, no. 11, pp. 2255–2261, Nov. 2003.

[10] M. S. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: Freeman, 1979.

[11] W. Rao, A. Orailoglu, and R. Karri, "Logic mapping in crossbar-based nanoarchitectures," *IEEE Des. Test Comput.*, vol. 26, no. 1, pp. 68–76, Jan./Feb. 2009.

[12] A. DeHon and H. Naeimi, "Seven strategies for tolerating highly defective fabrication," *IEEE Des. Test Comput.*, vol. 22, no. 4, pp. 306–315, Jul./Aug. 2005.

[13] Y. Yellambalase and M. Choi, "Cost-driven repair optimization of reconfigurable nanowire crossbar systems with clustered defects," *J. Syst. Archit.*, vol. 54, no. 8, pp. 729–741, Aug. 2008.

[14] M. O. Simsir, S. Cadambi, F. Ivančić, M. Roetteler, and N. K. Jha, "A hybrid nano-CMOS architecture for defect and fault tolerance," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 5, no. 3, Aug. 2009, Art. no. 14.

[15] S. Gören, H. F. Ugurdag, and O. Palaz, "Defect-aware nanocrossbar logic mapping through matrix canonization using two-dimensional radix sort," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 7, no. 3, Aug. 2011, Art. no. 12.
[16] C. Tunc and M. B. Tahoori, "Variation tolerant logic mapping for crossbar array nano architectures," in *Proc. 15th Asia South Pacific Design Autom. Conf.*, Jan. 2010, pp. 855–860.
[17] M. Zamani, H. Mirzaei, and M. B. Tahoori, "ILP formulations for variation/defect-tolerant logic mapping on crossbar nano-architectures," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 3, Sep. 2013, Art. no. 21.
[18] B. Yuan, X. Yao, B. Li, and T. Weise, "A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nanoarchitectures," *IEEE Trans. Evol. Comput.*, vol. 18, no. 6, pp. 846–859, Dec. 2014.
[19] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: Model, taxonomy, and design issues," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 474–488, Oct. 2005.
[20] X. Chen, Y.-S. Ong, M.-H. Lim, and K. C. Tan, "A multi-facet survey on memetic computation," *IEEE Trans. Evol. Comput.*, vol. 15, no. 5, pp. 591–607, Oct. 2011.
[21] F. Zhong, B. Yuan, and B. Li, "Hybridization of NSGA-II with greedy re-assignment for variation tolerant logic mapping on nano-scale crossbar architectures," in *Proc. Genet. Evol. Comput. Conf. Companion*, Jul. 2014, pp. 97–98.
[22] A. DeHon, "Nanowire-based programmable architectures," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 1, no. 2, pp. 109–162, Jul. 2005.
[23] T. Hogg and G. S. Snider, "Defect-tolerant adder circuits with nanoscale crossbars," *IEEE Trans. Nanotechnol.*, vol. 5, no. 2, pp. 97–100, Mar. 2006.
[24] T. Hogg and G. S. Snider, "Defect-tolerant logic with nanoscale crossbar circuits," *J. Electron. Test.*, vol. 23, nos. 2–3, pp. 117–129, 2007.
[25] J. Dai, L. Wang, and F. Jain, "Analysis of defect tolerance in molecular crossbar electronics," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 4, pp. 529–540, Apr. 2009.
[26] M. Crocker, X. S. Hu, and M. Niemier, "Defects and faults in QCA-based PLAs," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 5, no. 2, Jul. 2009, Art. no. 8.
[27] M. B. Tahoori, "Application-independent defect tolerance of reconfigurable nanoarchitectures," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 2, no. 3, pp. 197–218, Jul. 2006.
[28] B. Yuan and B. Li, "A fast extraction algorithm for defect-free subcrossbar in nanoelectronic crossbar," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 10, no. 3, Apr. 2014, Art. no. 25.
[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2001.
[30] P. Merz and B. Freisleben, "Fitness landscape analysis and memetic algorithms for the quadratic assignment problem," *IEEE Trans. Evol. Comput.*, vol. 4, no. 4, pp. 337–352, Nov. 2000.
[31] S. Salcedo-Sanz and X. Yao, "A hybrid Hopfield network-genetic algorithm approach for the terminal assignment problem," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 6, pp. 2343–2353, Dec. 2004.
[32] S. Salcedo-Sanz, Y. Xu, and X. Yao, "Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems," *Comput. Oper. Res.*, vol. 33, no. 3, pp. 820–835, 2006.
[33] H. C. W. Lau, T. M. Chan, and W. T. Tsui, "Item-location assignment using fuzzy logic guided genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 765–780, Dec. 2008.
[34] C. F. T. Soares, A. C. de Mesquita Filho, and A. Petraglia, "Optimizing capacitance ratio assignment for low-sensitivity SC filter implementation," *IEEE Trans. Evol. Comput.*, vol. 14, no. 3, pp. 375–380, Jun. 2010.
[35] Y. Su and W. Rao, "Runtime analysis for defect-tolerant logic mapping on nanoscale crossbar architectures," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit.*, Jul. 2009, pp. 75–78.
[36] J. He, X. Yao, and J. Li, "A comparative study of three evolutionary algorithms incorporating different amounts of domain knowledge for node covering problem," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 35, no. 2, pp. 266–271, May 2005.
[37] R. L. Iman and J. M. Davenport, "Approximations of the critical region of the fbietkan statistic," *Commun. Statist.-Theory Methods*, vol. 9, no. 6, pp. 571–595, 1980.
[38] O. J. Dunn, "Multiple comparisons among means," *J. Amer. Statist. Assoc.*, vol. 56, no. 293, pp. 52–64, 1961.
[39] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Jan. 2006.

Bo Yuan (M'15) received the B.Sc. and Ph.D. degrees in electronic information science and technology from the University of Science and Technology of China (USTC), Hefei, China, in 2009 and 2014, respectively.

He is currently a Post-Doctoral Fellow with the USTC–Birmingham Joint Research Institute in Intelligent Computation and Its Applications, School of Computer Science and Technology, USTC. His current research interests include evolutionary computation, electronic design automation, and machine learning.

Bin Li (M'07) received the B.Sc. degree from the Hefei University of Technology, Hefei, China, in 1992, the M.Sc. degree from the Institute of Plasma Physics, Chinese Academy of Sciences, Hefei, in 1995, and the Ph.D. degree from the University of Science and Technology of China (USTC), Hefei, in 2001.

He is currently a Professor with the School of Information Science and Technology, USTC. He has authored or co-authored over 40 refereed publications. His current research interests include evolutionary computation, memetic algorithms, pattern recognition, and real-world applications.

Dr. Li is the Founding Chair of the IEEE Computational Intelligence Society Hefei Chapter, a Counselor of the IEEE USTC Student Branch, a Senior Member of the Chinese Institute of Electronics (CIE), and a member of the Technical Committee of the Electronic Circuits and Systems Section of CIE.

Huanhuan Chen (SM'15) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 2004 and the Ph.D. degree in computer science from the University of Birmingham, Birmingham, U.K., in 2008.

He is currently a Full Professor with the UBRI, School of Computer Science and Technology, USTC. His current research interests include statistical machine learning, data mining, fault diagnosis, and evolutionary computation.

Dr. Chen received the 2015 International Neural Network Society (INNS) Young Investigator Award, the 2012 IEEE Computational Intelligence Society Outstanding Ph.D. Dissertation Award (the only winner), and the 2009 CPHC/British Computer Society Distinguished Dissertations Award (the runner up). His work Probabilistic Classification Vector Machines on Bayesian machine learning received the IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award (bestowed in 2011 and only one paper in 2009). He is currently the Associate Editor of the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS.

Xin Yao (F'03) is currently a Chair (Professor) of Computer Science and the Director of the Centre of Excellence for Research in Computational Intelligence and Applications with the University of Birmingham, Birmingham, U.K. He has been invited to give more than 70 keynote and plenary speeches at international conferences. He has authored over 400 refereed publications in international journals and conferences. His current research interests include evolutionary computation and ensemble learning.

Prof. Yao received the 2001 IEEE Donald G. Fink Prize Paper Award, the 2010 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award, the 2010 BT Gordon Radley Award for Best Author of Innovation (Finalist), the 2011 IEEE TRANSACTIONS ON NEURAL NETWORKS Outstanding Paper Award, and many other best paper awards. He also received the Prestigious Royal Society Wolfson Research Merit Award in 2012 and the IEEE Computational Intelligence Society (CIS) Evolutionary Computation Pioneer Award in 2013. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2003 to 2008. He serves as the President of the IEEE CIS.